NPS-MV-02-001

NAVAL POSTGRADUATE SCHOOL
Monterey, California

An Investigation of Alternative Algorithms for Singularity–Free Estimation of Rigid Body Orientation from Earth Gravity and Magnetic Field Measurements

by

Robert B. McGhee
Eric R. Bachmann
Xiaoping Yun
Michael J. Zyda

October 2001

Prepared for: NPS MOVES Institute

REPORT DOCUMENTATION PAGE

Form approved

OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 2001	Technical Report

4. TITLE AND SUBTITLE
An Investigation of Alternative Algorithms for Singularity–Free Estimation of Rigid Body Orientation from Earth Gravity and Magnetic Field Measurements

5. FUNDING
N6M ARO

6. AUTHOR(S)
McGhee, Robert B., Bachmann, Eric R., Yun, Xiaoping, and Zyda, Michael J.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5118

8. PERFORMING ORGANIZATION REPORT NUMBER
NPS-MV-02-001

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words.)
Recently, a new impetus has been given to work to produce a whole body human/computer interface system, which encumbers the wearer to a minimum degree and operates over long distances. The input portion of such an interface can be derived by appropriate processing of signals from a nine-axis sensor package consisting of a three-axis angular rate sensor, a three-axis magnetometer, and a three-axis linear accelerometer. This paper focuses on data processing algorithms for MARG sensors. Since human limb segments are capable of arbitrary motion, Euler angles do not provide an appropriate means for specifying orientation. Instead, quaternions are used for this purpose. Since a quaternion is a four-dimensional vector, and a MARG sensor produces nine signals, this data processing problem is "overspecified". This fact can be used to discriminate against sensor noise and to reduce the effects of linear acceleration on measurement of the gravity vector by accelerometers. Detailed computer simulation studies are used to confirm the effectiveness of the quaternion body-tracking filter.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Inertial Tracking, Magnetic Tracking, MARG Sensors, Complementary Filtering, Quaternions	32
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassfied	UL

NSN 7540-01-280-5800

Standard Form 298 Rev. 2-89)
Prescribed by ANSI Std 239-18

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

RADM David R. Ellison, USN
Superintendent

Richard Elster
Provost

This report was prepared for __NPS_MOVES Institute_____
and funded by _Army Research Office (ARO), Modeling and Simulation Office (N6M)

This report was prepared by:

Robert B. McGhee
Professor

Eric R. Bachmann
Research Assistant Professor

Xiaoping Yun
Professor

Michael J. Zyda
Professor

Reviewed by:

Michael J. Zyda
Director, MOVES Institute

Released by:

D. W. Netzer
Associate Provost and
Dean of Research

An Investigation of Alternative Algorithms for Singularity–Free Estimation of Rigid Body Orientation from Earth Gravity and Magnetic Field Measurements

R.B. McGhee, E.R. Bachmann, X.P. Yun, and M.J. Zyda
Naval Postgraduate School
Monterey, California
93943-5118 USA

1. Introduction

For millennia, it has been recognized that the orientation of a rigid body relative to the local vertical (gravity vector) can be specified by two angles. Since the Nineteenth Century, these angles have commonly been called "Euler" angles, usually designated by the reserved words "bank" (or "roll") and "elevation" (or "pitch") angles. Also since ancient times, it has been understood that a third angle, representing rotation about a vertical axis and usually called "azimuth" (or "heading"), is needed to completely specify rigid body orientation. This angle is by convention referenced to the local north vector and determined by sightings on the North Star or by suitably corrected compass readings.

With the invention and rapid proliferation of aircraft early in the Twentieth Century, some problems with the above approach to rigid body orientation soon became evident. Specifically, although vertical gyros were introduced to maintain a stable "artificial horizon" for measurement of aircraft elevation (or "climb") and bank angles, heading is undefined for an aircraft in a vertical orientation. While pilots are able to deal with such a "singularity" by simply ignoring instrument readings for a brief period of time while passing through the vertical, attempts to create ground based "flight simulators" encountered more fundamental problems with the use of Euler angles for orientation. In particular, it is now well understood that the non-orthogonal "body rotation rate to Euler angle rate" transformation matrix is singular in a vertical orientation (McGhee et al., 2000b). This can result in erratic results or even floating point overflow conditions in digital simulation of flight dynamics. Fortunately, this situation is mathematical in nature, and not physical. Thus, by adopting a quaternion representation of orientation in place of Euler angles, this singularity problem is completely eliminated while computational complexity is at the same time greatly reduced (Cooke et al., 1992).

While the advantages of quaternions over Euler angles for non-singular orientation representation are now generally appreciated with regard to computer simulation of rigid body dynamics, this is not the case in orientation sensing. Specifically, while low cost and effective orientation sensors based on three-axis magnetometers and three-axis accelerometers are now available, most "electronic compasses" and related systems still use computations based on Euler angles. (Frey, 1996; Yun et al., 1999; Precision Navigation, 1995; Foxlin, 1996) and are therefore unable to track orientation through vertical motion. While this is not important in applications such as ship and land vehicle navigation, it impacts severely attempts to use such sensors in other important areas such as aerial vehicle navigation (Gebre-Egziabher et al., 2000) and human limb segment motion tracking (McGhee et al, 2000a; Bachmann, 2000). Consequently, the purpose of this paper is to present a number of alternative algorithms for singularity-free orientation estimation based on three-axis sensing of Earth gravitational and magnetic fields. Both static and dynamic estimation based on such *sourceless* sensing are considered, and the

relative advantages and disadvantages of several algorithms are presented by means of mathematical analysis and computer simulation results.

2. Mathematical Formulation of Problem

In the absence of linear acceleration, an orthogonal triad of accelerometers senses the local gravity vector. Specifically, if q is the (non zero) orientation quaternion associated with a rigid body, m is a constant unit vector (quaternion with zero first element) pointing in the local down direction, and $\vec{y}_g$ is the corresponding unit vector determined from the outputs of an accelerometer triad, then, assuming accelerometer axes coincide with body axes and that there is no error in the accelerometer outputs, it follows that (McGhee et al., 2000a):

$$\vec{y}_g(q) = Ve\left(q^{-1} \otimes m \otimes q\right) \tag{1}$$

In this expression, Ve designates the *vector part* (last three components) of a quaternion and $\otimes$ is the quaternion product. Likewise, if $\vec{y}_n$ is the output of a perfect three-axis magnetometer, and n is a unit vector aligned with the local Earth magnetic field vector, then

$$\vec{y}_n(q) = Ve\left(q^{-1} \otimes n \otimes q\right) \tag{2}$$

With these definitions, *the computed measurement vector*, $\vec{y}(q)$, can be defined as the 6x1 column vector

$$\vec{y}(q) = \left(\vec{y}_g(q),\, \vec{y}_n(q)\right)^T \tag{3}$$

where the comma denotes concatenation. If the actual measurement vector, $\vec{y}_0$, is defined as the transpose of the concatenation of the unit row vector obtained by normalizing the real physical output of an accelerometer triad and the corresponding unit vector from a magnetometer triad, then the modeling error vector, $\vec{\varepsilon}(q)$, is defined by

$$\vec{\varepsilon}(q) = \vec{y}_0 - \vec{y}(q) \tag{4}$$

The problem of obtaining an *estimated orientation quaternion*, $\hat{q}$, from the measurement vector, $\vec{y}_0$, can thus be viewed as the mathematical problem of finding a q that in some sense minimizes $\vec{\varepsilon}(q)$.

3. Least Squares Estimation and Gauss-Newton Iteration

Examination of Eq. (4) reveals that this relationship amounts to six equations involving the four unknown components of $\hat{q}$, the value of q that minimizes $\vec{\varepsilon}$. This observation reveals two difficulties. First of all, if one were to simply equate $\vec{\varepsilon}(q)$ to a 6x1 zero vector, the corresponding value for q is *overspecified* since, in general, four unknowns can satisfy only four equations. The usual way to address this kind of problem

is to define a scalar *criterion function* on $\vec{\varepsilon}$ that is minimized by $\hat{q}$. The most common choice for such a function is the *squared error* criterion function,

$$h(q) = \vec{\varepsilon}(q)^T \vec{\varepsilon}(q) = \left|\vec{\varepsilon}(q)\right|^2 \tag{5}$$

In the absence of measurement and modeling errors, $\hat{q}$ will reduce $h(q)$ to zero, but in any practical case, the minimum value for this function will be a positive number. Such a minimizing q is called the *least squares estimate* for the true (unknown) value of q (McGhee, 1967).

While the above approach solves the overspecification problem, there is a hidden *underspecification* problem associated with minimizing Eq. (5). Specifically, if $q = \hat{q}$ is a minimizing value for q, then as shown in Appendix B, $q = \alpha\hat{q}$ is also a minimizing value, where α is any non-zero scalar. This means that any attempt to find $\hat{q}$ through iterative linearization of $\vec{y}(q)$ will involve a singular matrix. It is important to realize that this is not a theoretical problem, but rather a basic fact relating to the most common way of solving equations such as Eq. (5), called *Gauss-Newton* iteration. Specifically, let the 6 x 4 matrix X be defined by:

$$X_{ij} = \frac{\partial y_i(q)}{\partial q_j} \tag{6}$$

Then, the Gauss-Newton iteration equation that computes the change in q needed to minimize $h(q)$ is (McGhee et al., 2000a):

$$\Delta q = \left[X^T X\right]^{-1} X^T \vec{\varepsilon}(q) \tag{7}$$

$$= S^{-1} X^T \vec{\varepsilon}(q) \tag{8}$$

However, because X is not of full rank, the *regression matrix*, S, is singular and Δq cannot in fact be determined from this equation.

One way to deal with the non-uniqueness of orientation quaternions is to restrict them to be unit quaternions. The most obvious way to do this is to add to $h(q)$ some function, $g(q)$, which is continuous and differentiable for all q and positive for all values of q other than unit quaternions. One such suitable function is

$$g(q) = \left(|q|^2 - 1\right)^2 \tag{9}$$

Evidently, if

$$\varphi(q) = h(q) + g(q) \tag{10}$$

then minimizing $\varphi(q)$ yields a $\hat{q}$ that is a unique unit quaternion except for a sign ambiguity. That is, if the unit quaternion $\hat{q}$ minimizes $\varphi(q)$, then so does $-\hat{q}$.

Fortunately, since q and $-q$ accomplish the same rotation, this degree of non-uniqueness does not disturb numerical minimization techniques such as Gauss-Newton iteration (McGhee et al., 2000a). Thus, utilization of Eq. (10) provides one means of obtaining a non-singular estimate of the orientation quaternion for a rigid body from measurements of Earth gravity and magnetic field vectors. While this is the most obvious way to solve this problem, the authors have found better solutions as described in the following paragraphs, and therefore do not advocate this "naïve" method. That is, in the remainder of this paper, the function $g(q)$ in Eq. (10) will be omitted and restriction of $\hat{q}$ to unit vectors will be accomplished by more effective means.

4. Modified Computed Measurement Function

An alternative approach to dealing with the above singularity problem is to redefine the computed measurement functions given by Eq. (1) and (2) by replacing the inverse of q by its *conjugate* defined as (McGhee et al., 2000a):

$$q^* = \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \end{pmatrix} \tag{11}$$

With this substitution, using the product rule of differential calculus, it follows that (Henault, 1997; McGhee et al., 2000a)

$$\frac{\partial \vec{y}}{\partial q_0} = \begin{pmatrix} m \otimes q + q^* \otimes m, & n \otimes q + q^* \otimes n \end{pmatrix}^T \tag{12}$$

$$\frac{\partial \vec{y}}{\partial q_1} = \begin{pmatrix} -i \otimes m \otimes q + q^* \otimes m \otimes i, & -i \otimes n \otimes q + q^* \otimes n \otimes i \end{pmatrix}^T \tag{13}$$

$$\frac{\partial \vec{y}}{\partial q_2} = \begin{pmatrix} -j \otimes m \otimes q + q^* \otimes m \otimes j, & -j \otimes n \otimes q + q^* \otimes n \otimes j \end{pmatrix}^T \tag{14}$$

$$\frac{\partial \vec{y}}{\partial q_3} = \begin{pmatrix} -k \otimes m \otimes q + q^* \otimes m \otimes k, & -k \otimes n \otimes q + q^* \otimes n \otimes k \end{pmatrix}^T \tag{15}$$

In these equations, the symbols i, j, and k are unit vectors (in quaternion form) pointing in the local Earth-fixed x, y, and z directions (usually, north, east, and down, respectively). Also, it is to be understood that only the vector part of each triple quaternion product is used.

Of course, these results are different (and much simpler) than those obtained from differentiation of Eq. (1) and (2) (McGhee, et al., 2000a). The question is, how is this change justified? The answer is that *providing that q is a unit quaternion*, its inverse is just its conjugate. Thus, if the iterative calculation of $\hat{q}$ involves *normalization* to a unit quaternion on every cycle (by dividing $q + \Delta q$ by its magnitude), then the above simplified derivatives should function correctly in Eq. (8), and should result in rapid convergence to a correct value for $\hat{q}$, without any singularity problems. Simulation experiments have shown this to be true (Henault, 1997; McGhee et al., 2000a). Moreover,

these equations have been used with good results in an innovative real-time human motion tracking system (Bachmann, 2000; Bachmann et al., 2001).

5. Reduced Order Estimation

Although the authors believe that the use of Eq. (12)-(15) in (Henault, 1997) represents the first effective means found for singularity-free "sourceless" rigid body attitude estimation, we recognize that, in fact, the components of a unit quaternion are not independent variables. Rather, up to a sign ambiguity, q_0, the *real part* of a unit quaternion, can be determined from $\begin{pmatrix} q_1 & q_2 & q_3 \end{pmatrix}$, the vector part of the quaternion, from the relation:

$$q_0^{\,2} + q_1^{\,2} + q_2^{\,2} + q_3^{\,2} = 1 \tag{16}$$

This being the case, there should be a 3x3 regression matrix version of the Gauss-Newton iteration relation of Eq. (8). In (McGhee et al., 2000a), we were able to obtain this problem reduction by observing that if $\hat{q}$ and $\hat{q} + \Delta q$ are both unit vectors, then in the limit, as Δq approaches zero, it must be orthogonal to q. Motivated by this observation, we derived the *orthogonal quaternion theorem* that states that if p is a quaternion orthogonal to another quaternion, q, then

$$p = q \otimes v \tag{17}$$

where v is the unique vector

$$v = \begin{pmatrix} 0 & v_1 & v_2 & v_3 \end{pmatrix} = q^{-1} \otimes p \tag{18}$$

Thus, determining a value for v that minimizes $\varphi(q)$ determines $\hat{q}$ (McGhee et al., 2000a).

While the results of combining Eq. (12)-(15) with the above problem reduction produced the best algorithm we have actually used to date in physical experiments (Bachmann, 2000), we have recently become aware of a better approach to the 3 x 3 problem formulation that leads both to simpler equations and to deeper mathematical insights (Gebre-Egziabher et al., 2000). Specifically, from Appendix B, let q_r be the *incremental rotation quaternion* given by:

$$q_r = \begin{pmatrix} 1 & r_1 & r_2 & r_3 \end{pmatrix} = \begin{pmatrix} 1, & \vec{r} \end{pmatrix} \tag{19}$$

Evidently, as the *rotation vector* $\vec{r}$ approaches zero, then q_r approaches a unit quaternion. Thus, if a value is found for q_r that reduces the squared error criterion function, $\varphi(q)$, it follows that

$$\hat{q}_{new} = \hat{q}_{old} \otimes q_r \tag{20}$$

and that, in the limit, as $\vec{r}$ approaches zero, $\hat{q}_{new}$ will also be a unit quaternion. More practically, the results of Eq. (20) should be normalized to a unit quaternion after every

iterative application of this equation (McGhee et al., 2000a), thereby removing any limitations on the magnitude of $\vec{r}$.
Eq. (20) can be written in additive form by noting that

$$\Delta q = \hat{q}_{new} - \hat{q}_{old} = \hat{q}_{old} \otimes \begin{pmatrix} 1 & r_1 & r_2 & r_3 \end{pmatrix} - \hat{q}_{old} \otimes \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} \tag{21}$$

$$= \hat{q}_{old} \otimes \begin{pmatrix} 0 & r_1 & r_2 & r_3 \end{pmatrix} \tag{22}$$

where

$$\vec{r}^{\,T} = \left[X^T X \right]^{-1} X^T \vec{\varepsilon}\left(\hat{q}_{old}\right) \tag{23}$$

and, from Appendix A, X is the 6x3 *reduced order* matrix:

$$X = 2 \begin{bmatrix} 0 & -y_3 & y_2 \\ y_3 & 0 & -y_1 \\ -y_2 & y_1 & 0 \\ 0 & -y_6 & y_5 \\ y_6 & 0 & -y_4 \\ -y_5 & y_4 & 0 \end{bmatrix} \tag{24}$$

To understand these results, it is important to recognize that the elements of X are just the components of the computed measurement vector, $\vec{y}\left(\hat{q}_{old}\right)$, given by Eq. (1)-(3). Since this vector is needed in every cycle of Gauss-Newton iteration to compute the modeling error vector, $\vec{\varepsilon}\left(\hat{q}_{old}\right)$, given by Eq. (4), it follows that the above value for X is "free" since all terms are known once $\vec{\varepsilon}\left(\hat{q}_{old}\right)$ has been computed. While the numerical value for X obtained in this way is identical to the value obtained using the orthogonal quaternion theorem (McGhee et al., 2000a), Eq. (24) provides an order of magnitude reduction in the amount of computation needed to obtain X, and is therefore of significant importance in applying Gauss-Newton iteration in practical real-time orientation tracking systems.

6. Convergence and Accuracy of Gauss Newton Iteration

While necessary and sufficient conditions for local convergence of Gauss-Newton iteration are known, these require evaluation of the eigenvalues of a rather complex matrix (McGhee, 1963). As a practical matter, it is generally more effective to investigate convergence by means of a simulation study. Moreover, when this is done, not only convergence, but also the effect of sensor noise on estimation accuracy can be examined. With this in mind, a computer simulation of the static estimation problem was written in ANSII Common Lisp (Appendix C). In this simulation, computer generated gaussian white noise samples were added to each of the six components of a computed measurement vector obtained by using Eq. (3). The noise standard deviation, σ_n, was the same for all components of this vector, and each six dimensional noise sample was independently generated for every simulated measurement vector. The true orientation

quaternion, q_{true}, needed by Eq. (3) was also randomly generated, by using independent uniformly distributed random variables in the interval [-1 1] for each of the four components of this quaternion, and then normalizing the results to a unit quaternion.

Following the generation of artificial data using the above procedure, in the simulation study, Eq. (22)-(24) were iteratively applied a specified number of times. The starting value for each such cycle of iterations was obtained by adding to each component of the true orientation quaternion a random number uniformly distributed in the interval [-0.1, 0.1], and then normalizing this result to obtain a unit quaternion. Table 1 shows a typical result obtained in such a trial in the case of perfect data ($\sigma_n = 0$). As can be seen from this table, in these circumstances, almost all of the error in estimating the true value for q is removed in the first cycle of Gauss-Newton iteration.

	Estimated Orientation Quaternion Components			
	q_0	q_1	q_2	q_3
Initial Estimate	-0.19242062	0.5861363	-0.52690697	-0.58462614
Iteration 1	-0.16005535	0.64316887	-0.53970546	-0.5190704
Iteration 2	-0.15987878	0.6468387	-0.5358498	-0.518559
Iteration 3	-0.15988407	0.6468314	-0.5358464	-0.51857
True	-0.15988402	0.6468314	-0.53584635	-0.51856995

Table 1: Typical Sequence of Values for Orientation Quaternion Estimate using Gauss-Newton Iteration with Noiseless Data

To evaluate the effects of measurement noise on the convergence and accuracy of Gauss-Newton iteration, the experiment described above was repeated with artificial measurement data to which gaussian noise with a standard deviation of 0.01 was added to each of the six data components followed by normalization of both the simulated noisy accelerometer data and magnetometer data. Typical results are shown in Table 2 below. As can be seen, even though the estimation sequence still converges very quickly, the final result differs from the true value for q by a small amount, on the same order as the error in the measurement vector. This is, of course, to be expected since part of the significance of the results of Table 1 is that Gauss-Newton linearization of the dependence of estimation error on measurement error (as represented by Eq. (7)) is quite accurate.

	Estimated Orientation Quaternion Components			
	q_0	q_1	q_2	q_3
Initial Estimate	0.12020314	0.2773305	0.92554724	-0.22803806
Iteration 1	0.13213184	0.34180763	0.9060598	-0.21157604
Iteration 2	0.13303357	0.34058997	0.90590346	-0.21363427
Iteration 3	0.13302392	0.34063232	0.90588933	-0.21363264
True	0.13386706	0.33879966	0.9073919	-0.20960511

Table 2: Typical Sequence of Values for Orientation Quaternion Estimate using Gauss-Newton Iteration with Noise Standard Deviation Equal to 0.01 on Each Component of Accelerometer and Magnetometer Output

While the above Table 1 and Table 2 give a general idea of the convergence and accuracy properties of Gauss-Newton iteration, in order to deal with a much larger

sample, a series of simulation trials was conducted in which each experiment was repeated one thousand times. For each such trial, the root mean square (RMS) quaternion estimation error was computed and recorded for each successive cycle of Gauss-Newton iteration. More specifically, each trial produced a series of estimated values such as those recorded in Table 1 and Table 2. For each estimate in such a series, the *squared error function* is simply the square of the length of the difference between the true and the estimated orientation quaternion. The RMS error is then just the square root of the average of the squared error function (at each level of iteration) over the 1000 trials for each noise level selected. Table 3 summarizes typical results for such an experiment.

Noise Level	**RMS Error in Estimated Quaternion**				
	Initial	**Iteration1**	**Iteration 2**	**Iteration 3**	**Iteration 4**
0.00	0.10071685	0.0051641823	1.7919934e-5	8.850226e-8	8.274388e-8
0.01	0.098774455	0.015352701	0.014763316	0.014770619	0.014770887
0.03	0.09863774	0.04265576	0.043464825	0.043680053	0.043695707
0.10	0.100237824	0.13531718	0.13646685	0.14414792	0.14421917

Table 3: RMS Error in Estimated Orientation Quaternion Averaged Over 1000 Trials

In this table, "noise level" refers to the standard deviation of the gaussian noise added to each component of the simulated measurement vector. The true value for the orientation vector was randomly generated using the same technique as in Table 1 and Table 2 above.

There are several important conclusions to be drawn from the results presented in Table 3. First of all, for noiseless data, there is no significant improvement in estimation accuracy after three cycles of Gauss-Newton iteration. For all other levels of noise considered, one cycle of Gauss-Newton iteration appears to be sufficient. These results show that the linearization of Eq. (7) is very accurate, especially for data containing realistic amounts of measurement noise. Closer examination of the entries in this table reveals the apparently anomalous result that, for $\sigma_n = 0.10$, RMS estimation error actually increases on each iteration cycle. However, this is an artifact resulting from the fact that, as described above, the amount of artificially induced starting error is the same for all measurement noise levels, and for high noise levels is less than that resulting from the actual minimization of the squared error criterion function of Eq. (5). Finally, there is the quantitative result that, for all levels of gaussian measurement noise considered, the RMS value of the length of the difference between the true and estimated orientation quaternion is about 45% more than the per component measurement error. This provides a useful "rule of thumb" for relating sensor noise to orientation estimation errors, and is further evidence of the accuracy of the linearization accomplished by Gauss-Newton iteration.

While all of the above results were obtained by recalculating the X matrix on every cycle of iteration, further experiments have shown that this is not needed to obtain the results shown in Table 3. Moreover, it has also been found that the X matrix of Eq. (24) can be computed directly from the components of the noisy measurement vector, $\vec{y}_0$, and good convergence still results. These experimental findings are without theoretical justification at present.

7. Dynamic Estimation

All of the above discussion relates to the *static estimation* problem in which only a single measurement vector, $\vec{y}_0$, is available. More typically, in real orientation estimation problems, a time sequence of measurement vectors will be available, leading to an *orientation tracking* problem in which each new measurement vector is used to obtain a new orientation estimate. In the special case that it is known *a priori* that the object being tracked is in fact stationary, one approach to tracking would be to simply average all data vectors up to the last one, and then estimate the orientation quaternion from this average using the methods described above. However, a more interesting and challenging problem is to suppose that the object being tracked is moving in an unknown way, and that some type of *filter* is used to track such motion while discriminating against measurement noise and the confounding effect of linear acceleration on the measurement of gravity by an accelerometer triad. One form for such a filter is shown in Figure 1 below.

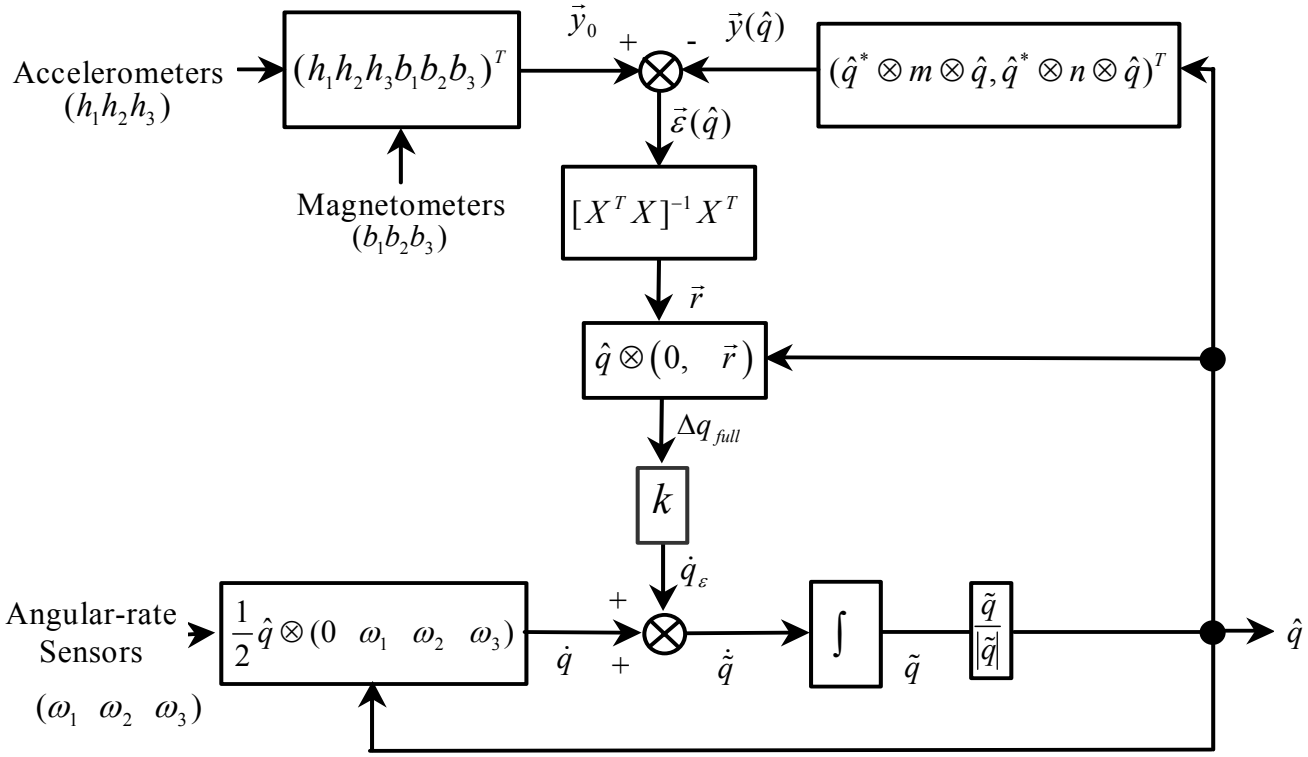

Figure 1: Quaternion-Based Orientation Estimation Filter

Examination of Figure 1 shows the incorporation of a *feedback loop* employing reduced order Gauss-Newton iteration. Specifically, on this figure, Δq_{full} is just the result of one iteration of Eq. (21). Ignoring for the moment the input from the angular rate sensors indicated on this diagram, it can be seen that Δq_{full} is multiplied by a scalar gain factor, k, integrated, normalized, and the result then used to iterate again on Eq. (21). This process is the same as described in the discussion relating to Tables 1 and 2 above with two important differences as follows. First of all, in static estimation, the

measurement vector $\vec{y}_0$ is treated as a constant, while in orientation tracking, a new value for $\vec{y}_0$ is available on every cycle of iteration. Secondly, the functioning of the integrator together with the gain factor multiplication has yet to be explained. This discussion follows.

If the X matrix were constant (which it is not in realistic tracking problems), then the feedback loop of Figure 1 could be analyzed in continuous time using the linearizing assumption that Δq_{full} represents the true error in $\hat{q}$. Such an analysis reveals that this feedback loop functions as a low pass filter, with time constant equal to $1/k$ (Bachmann et al., 1999). The function of this filter is to discriminate against the effects of sensor noise and linear acceleration in circumstances when such signals are of relatively high frequency compared to actual object rotational motion (Bachmann et al., 1999). To gain some understanding of this idea, consider the (idealized) special case in which successive values of $\vec{y}_0$ do not change (stationary object, noiseless sensors). Then, ignoring the nonlinear normalization operation, and assuming Δq_{full} is exact, from Figure 1,

$$\dot{\hat{q}} = k \Delta q_{full} \tag{25}$$

If q_{true} is the actual true value for the orientation quaternion associated with $\vec{y}_0$, then under the above assumptions,

$$\Delta q_{full} = q_{true} - \hat{q} \tag{26}$$

Thus, combining these two equations,

$$\dot{\hat{q}} + k\,\hat{q} = k\,q_{true} \tag{27}$$

By any of a number of well known methods (Kuo, 1995), the solution to this differential equation is easily shown to lead to the result

$$\Delta q_{full}(t) = \Delta q_{full}(0)\; e^{-kt} \tag{28}$$

That is, in this special case, the error in $\hat{q}$ exponentially declines to zero.

In fact, the above result is just a limiting value as the discrete time filter loop *cycle time*, Δt, approaches zero and is included here only to show how continuous time analysis can give some insight into the functioning of discrete time (digital computer) feedback systems. In reality, in digital systems, the integration of Figure 1 is most often replaced by a simple summation. That is, after n cycles of iteration,

$$\tilde{q}(n\Delta t) = \tilde{q}(0) + \sum_{i=1}^{n}\left[\left(\dot{\tilde{q}}(i\Delta t)\right)\Delta t\right] \tag{29}$$

Evidently, for each step in this summation, under the simplifying assumptions of the above analysis, $\tilde{q}$ is incremented by an amount

$$\Delta\tilde{q} = k\,\Delta t\,\Delta q_{full} \tag{30}$$

This relationship is key to understanding the functioning of an implementation of Figure 1 on a digital computer utilizing a fixed cycle time, Δt. Specifically, this result shows that if $k\,\Delta t = 1$, then the entire Gauss-Newton correction step will be performed on just one iteration cycle rather than in the exponential fashion of a continuous time filter (as determined by Eq. (28)). For $0 < k < 1/\Delta t$, on the first cycle, the fraction of Δq_{full} corrected is $k\,\Delta t$, so the remaining error is $(1 - k\,\Delta t)\,\Delta q_{full}$. This action will be repeated on the next cycle so that after n cycles

$$\Delta q(n) = q_{true} - \tilde{q}(n) = \Delta q_{full}(0)(1 - k\Delta t)^n \tag{31}$$

This analysis also applies to larger values of k, from which Eq. (31) reveals that for such values the error in $\tilde{q}$ oscillates in sign (that is, there is a degree of *overcorrection* in tracking) which diverges for $k\,\Delta t > 2$. Finally, since Eq. (31) uses just the first two terms in the Taylor series for $e^{-k\Delta t}$, as is well known in linear feedback control theory (Kuo, 1995), it follows that the results of this equation converge to Eq. (28) as Δt approaches zero.

The purpose of the above analysis is to give some insight into the functioning of one approach to dynamic estimation of the orientation quaternion of a rigid body, and also to expose some of the basic effects of time sampling on discrete time filters. In reality, this is a very deep subject with a well developed theory that goes far beyond the scope of this paper (Kuo, 1995). Perhaps the main point to be made in conclusion to this section of this paper is that numerous simulation and real time experiments have shown that the linearization provided by Gauss-Newton iteration is so powerful that all of the above analysis accurately describes the time behavior of the nonlinear filter shown in Figure 1 (Bachmann, 2000; Bachmann et al. 1999). The next section of this paper explains the benefit of including rate sensor inputs as shown on this figure, and also briefly discusses "optimal" attitude estimation filters for cases in which full statistical models are available both for object motion and for measurement errors.

8. Complementary Filtering and Optimal Estimation

Turning again to Figure 1, suppose $k = 0$. In this situation, evidently $\dot{\tilde{q}} = \dot{q}$. Thus, in the (unrealistic) case that rate sensor data is perfect, then providing the integrator in this figure is correctly initialized it follows that, $\tilde{q} = \hat{q} = q_{true}$. With this approach, the time lag associated with estimation of orientation from accelerometer and magnetometer data (as discussed in the above Section 7 of this paper) is no longer present. While this method can in fact be used for some period of time (depending on the quality of the angular rate sensors), real angular rate data will be corrupted by many sources of inaccuracy including random noise, scale factor and bias errors, nonlinearities, etc. This being the case, every practical orientation tracker of the form of Figure 1 must use $k > 0$. Such a filter is called a *complementary* filter because the integration of angular rate data, which provides an accurate response to rapid changes in orientation, is *complemented* by the long term accuracy provided by the low pass filtering of accelerometer and magnetometer derived estimates of orientation using Gauss-Newton linearization (upper half of Figure 1).

While the above discussion is entirely qualitative, under assumptions of continuous time integration, perfect linearization by the Gauss-Newton procedure, and exact measurement data, it can be shown that the complementary filter of Figure 1 provides an errorless value for $\hat{q}$ for all non-negative values of k (Bachmann, 2000; Bachman et al., 1999). That is, for $k \geq 0$, the "quickening" of the filter response by the use of angular rate information is exactly complementary to the "drift correction" provided by the use of accelerometer and magnetometer data. However, based on the analysis of the above Section 7 of this paper, in a real digital implementation of this system, a value for k in the range $0 < k < 2/\Delta t$ must be used. While considerable theory exists to provide a basis for choosing a good value for k in a given set of circumstances (Brown and Hwang, 1997), such theory requires accurate knowledge of the statistical properties of both the *measurement* and *maneuver* processes associated with a given rigid body. Complementary filters are typically used when such statistics are lacking, so the usual approach to selecting a value for k is to "tweak" it in an experimental setting until satisfactory performance is achieved (Bachmann, 2000). One useful "rule of thumb" to guide in the selection of k is that for sinusoidal motion of a rigid body, the greatest weight is placed on angular rate data for frequencies higher than the *crossover frequency* (in Hz) given by (Brown and Hwang, 1997; Bachmann 2000):

$$f_c = \frac{k}{2\pi} \tag{32}$$

Of course, below this frequency, greater weight is placed on orientation estimates obtained from accelerometer and magnetometer measurements.

In the (rare) special circumstance that all statistical properties of both object motion (maneuver process) and measurement noise are known, a well developed theory called "Kalman Filtering" exists for "optimal state estimation" for linear systems (Brown and Hwang, 1997). This theory automatically allows for greater weighting of reliable data in comparison to less reliable data in determining state estimates. For the nonlinear orientation quaternion estimation problem considered in this paper, an extended form of such filtering is possible through local linearization. Such filtering can be based on either reduced order (3x3) Gauss-Newton iteration (Gebre-Egziabher, 2000) or on a full (4x4) iteration using unit quaternions (Marins et al., 2001). Experimental results published in these two references show good convergence of the extended filters. Also, as in the preceding parts of this paper, such filters can be constructed either with or without angular rate sensors. The advantages of using such sensors are the same as described above for complementary filters. The most serious drawback to the use of "optimal" filters is that they are highly "tuned" to the assumed problem statistics and are not subject to "eyeball" adjustment in an experimental situation as is the case with the single parameter k involved in the complementary filter of Figure 1. Nevertheless, the powerful linearization capabilities of Gauss-Newton linearization as demonstrated by Table 3 in this paper means that Kalman filtering represents an approach to orientation estimation which should be seriously investigated as an alternative to complementary filtering when process statistics are stable and known to sufficient accuracy.

9. Summary, Conclusions, and Further Research

This paper attempts to illuminate a variety of alternative approaches to the singularity free estimation of rigid body orientation from measurements of the Earth's gravitational and magnetic fields by sensors attached to the body. Toward this end, a quaternion based theory is developed and validated by simulation studies. The results obtained include reduction of problem dimensionality by the use of incremental rotation quaternions having only three variable components rather than the four variables associated with an arbitrary quaternion. For tracking problems, in which orientation changes over time, since Gauss-Newton linearization of the orientation estimation problem introduces a time lag, "quickening" of the tracking filter through use of angular rate information is also described. An alternative view of this modification is that magnetometer and accelerometer data provide a "drift correction" for the integration of angular rate data to obtain an orientation quaternion.

Perhaps the main result of this paper, supported by simulation studies, is that Gauss-Newton iteration provides a surprisingly accurate linearization of the orientation quaternion estimation problem. This enables both complementary filtering and Kalman filtering approaches to singularity free orientation tracking using a nine-axis sensor package elsewhere called a MARG (Magnetic field, Angular Rate, Gravity) sensor (Bachmann et al., 2001). For static problems or tracking problems in which an output time delay due to low pass filtering is not objectionable, rate sensors can be eliminated (Foxlin, 1996; Bachmann, 2001)

With advances in sensor technology, especially relative to possible automotive applications (Teegarden et al., 1998), smaller and less expensive MARG sensors are gradually becoming available. The smallest nine axis system known to the authors is about 2 cubic inches in volume, and has been used successfully in human limb tracking experiments for virtual reality applications (Bachmann et al, 2001). Smaller packages are under development as part of this ongoing project. A patent has been filed relative to the hardware and software used in this application of the theory of this paper.

It is the authors' view that the theory and algorithms we have presented in this paper enable a very wide range of orientation tracking systems to be realized in a cost effective and entirely practical way. The software simplification enabled by the results of Appendix A is huge in comparison to methods based on Euler angles, and should enable both cost reduction and performance improvement in electronic compasses and other orientation measuring systems using embedded microprocessors. We hope that our work will encourage the development of such systems.

We do not consider that the theory we have presented is complete. In particular, when only partial information concerning process or noise statistics is available, it should be possible to use some form of "weighted error" criterion function in a complementary filter so that less emphasis would be placed on less reliable measurements. While some work has been started in this direction (Bachmann et al., 2001), much remains to be done. Likewise, so far as the authors know, there are no simulation studies relating to the degradation of Kalman filter performance in attitude estimation applications when measurement error or process statistical models are incorrect. The performance of such "mistuned" optimal filters relative to complementary filters with one or more adjustable parameters is important to all applications of Gauss-Newton linearization, and is a subject we intend to pursue in our human body tracking research.

As a final remark, we note that all of the present paper is devoted to orientation tracking only. A complete articulated body tracking system must also determine the location of one reference point on one of the rigid bodies comprising such a system (Bachmann et al, 2001). Differential GPS may be satisfactory for some outdoor applications (for example, tracking motion of individual human beings in virtual reality systems), but indoor solutions are still needed (Hightower & Borriello, 2001; Feng, 1996). Clearly there are many reasons for wanting to track the location of individuals and objects inside buildings, and we believe that this paper provides approaches to orientation tracking which will work well with such future systems to provide the posture data needed to complete full tracking of articulated rigid body models of human beings, and possibly some types of robotic systems, for remote computer display.

10. Acknowledgements

The authors wish to acknowledge the support of the U. S. Army Research Office (ARO) under Proposal No. 40410-MA and the U. S. Navy Modeling and Simulation Management Office (N6M) for making this work possible.

11. References

Bachmann, E., McGhee, R., Yun, X., and Zyda, M. (2001). "Inertial and Magnetic Posture Tracking for Inserting Humans into Networked Virtual Environments," *ACM Symposium on Virtual Reality Software and Technology (VRST)*, November 15-17, Banff, Canada, pp. 9 - 16.

Bachmann, E., McGhee, R., Yun X. & Zyda, M. (2001). "Real-Time Tracking and Display of Human Limb Segment Motions Using Sourceless Sensors and a Quaternion-Based Filtering Algorithm - Part II: Implementation and Calibration," MOVES Academic Group Technical Report NPS-MV-01-003, Naval Postgraduate School, Monterey, CA. http://www.movesinstitute.org/TechReports/NPS-MV-01-003.pdf

Bachmann, E. (2000). *Inertial and Magnetic Angle Tracking of Limb Segments for Inserting Humans into Synthetic Environments*, Ph.D. dissertation, Naval Postgraduate School, Monterey, CA. (Available at http://www.users.muohio.edu/bachmann/)

Bachmann, E., Duman, I., Usta, U., McGhee, R., Yun, X., & Zyda, M. (1999). "Orientation tracking for Humans and Robots Using Inertial Sensors," *International Symposium on Computational Intelligence in Robotics & Automation (CIRA 99)*, Monterey, CA, pp.187-194.

Brown, R. & Hwang, P. (1997). *Introduction to Random Signals and Applied Kalman Filtering,* 3rd Edition, John Wiley and Sons, New York.

Cooke, J., Zyda, M., Pratt, D., & McGhee, R. (1992). "NPSNET: Flight Simulation Modeling Using Quaternions", *Presence: Teleoperators and Virtual Environments*, Vol.1, No. 4, MIT Press, pp. 404 – 420.

Feng, J. (1996). *An RF Head-Tracker*, Phase I Final Report, NAVAIR SBIR Topic No. N95-144.

Frey, W. (1996). *Application of Inertial Sensors and Flux-Gate Magnetometers to Real-Time Human Body Motion Capture*, Master's Thesis, Naval Postgraduate School, Monterey, CA.

Foxlin, E. (1996). "Inertial Head-Tracker Sensor Fusion by a Complementary Separate-Bias Kalman Filter," *Proceedings of VRAIS `96*, IEEE, pp. 185-194.

Gebre-Egziabher, D., Elkaim, G., Powell, J., & Parkinson, B. (2000). "A Gyro-Free Quaternion-Based Attitude Determination System Suitable for Implementation Using Low Cost Sensors," *Position Location and Navigation Symposium, IEEE 2000* , pp. 185 - 192

Henault, G. (1997). *A Computer Simulation Study and Computational Evaluation for a Quaternion Filter for Sourceless Tracking of Human Limb Segment Motion*, M.S. thesis, Naval Postgraduate School, Monterey, CA.

Hightower, J., & Borriello, G. (2001). "Location Systems for Ubiquitous Computing," *IEEE Computer*, Volume: 34 Issue: 8 , pp. 57 – 66.

Kuipers, J. (1998). *Quaternions and Rotation Sequences*, Princeton University Press, Inc., Princeton, NJ.

Kuo, E. (1995). *Automatic Control Systems*, Seventh Edition, Prentice Hall, Inc., Englewood Cliffs, NJ, 1995.

Marins, J., Yun, X., Bachmann, E., McGhee, R., and Zyda, M. (2001). "An Extended Kalman Filter for Quaternion-Based Orientation Estimation Using MARG Sensors," *2001 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* October 29-November 3, Maui, Hawaii, USA, pp. 2003 - 2011.

McGhee, R., Bachmann, E., Yun X. & Zyda, M. (2000a). "Real-Time Tracking and Display of Human Limb Segment Motions Using Sourceless Sensors and a Quaternion-Based Filtering Algorithm - Part I: Theory," MOVES Academic Group Technical Report NPS-MV-01-001, Naval Postgraduate School, Monterey, CA. http://www.movesinstitute.org/TechReports//NPS-MV-01-001.pdf

McGhee, R., Bachmann, E., Brutzman, D., & Zyda, M. (2000b) "Rigid Body Dynamics, Inertial Reference Frames, and Graphics Coordinate Systems: A Resolution of Conflicting Conventions and Terminology," MOVES Academic Group Technical Report NPS-MV-01-002, Naval Postgraduate School, Monterey, CA.

McGhee, R. (1967). "Some Parameter-Optimization Techniques", Chapter 4.8, *Digital Computer User's Handbook*, ed. by M. Klerer and G.A. Korn, pp. 234 – 255.

McGhee, R. (1963). *Identification of Nonlinear Dynamic Systems by Regression Analysis Methods*, Ph.D. Dissertation, University of Southern California.

Precision Navigation (1995). *TCM2 Electronic Compass Module User's Manual*, Precision Navigation Inc.

Teegarden D., Lorenz, G., & Neul, R. (1998). "How to Model and Simulate Microgyroscope Systems," IEEE Spectrum, July, pp. 66 – 75.

Yun, X., Bachmann E., Arslan, S., & McGhee, R. (1999). "Testing and Evaluation of an Integrated GPS/INS System for Small AUV Navigation", *IEEE Journal of Oceanic Engineering*, Vol. 24, No. 3, pp. 396 – 404.

Appendix A: Derivation of Simplified X Matrix

It is well known that rotation of 3-vectors by matrix multiplication offers an alternative to rotation by quaternion multiplication. Specifically, referring to Eq. (1) an equivalent relation is:

$$\vec{y}_g^T(q) = \begin{pmatrix} y_1 & y_2 & y_3 \end{pmatrix}^T = R(q)m \tag{A-1}$$

where

$$m = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T \tag{A-2}$$

and, providing q is a unit quaternion, (Cooke et al., 1992; Gebre-Egziabher et al., 2000)

$$R(q) = \begin{bmatrix} 1-2\left(q_2^2+q_3^2\right) & 2\left(q_1q_2+q_3q_0\right) & 2\left(q_1q_3-q_2q_0\right) \\ 2\left(q_1q_2-q_3q_0\right) & 1-2\left(q_1^2+q_3^2\right) & 2\left(q_2q_3+q_0q_1\right) \\ 2\left(q_1q_3+q_3q_0\right) & 2\left(q_2q_3-q_1q_0\right) & 1-2\left(q_1^2+q_2^2\right) \end{bmatrix} \tag{A-3}$$

If q is an "incremental rotation quaternion", as defined by Eq. (19), then, as $\vec{r}$ approaches the zero vector, the matrix R can be linearized to the *skew symmetric matrix* (Gebre-Egziabher et al., 2000):

$$R_\delta(\vec{r}) = \begin{bmatrix} 1 & 2r_3 & -2r_2 \\ -2r_3 & 1 & 2r_1 \\ 2r_2 & -2r_1 & 1 \end{bmatrix} \tag{A-4}$$

Evidently, if $\hat{q}_{new}$ is defined as in Eq. (20), then, for small $\vec{r}$:

$$\vec{y}_g(q+\Delta q) = R_\delta(\vec{r})\,\vec{y}_g(q) \tag{A-5}$$

In component form, this means that:

$$y_1(q+\Delta q) = y_1(q) + 2r_3 y_2(q) - 2r_2 y_3(q) \tag{A-6}$$

$$y_2(q+\Delta q) = y_2(q) - 2r_3 y_1(q) + 2r_1 y_3(q) \tag{A-7}$$

$$y_3(q+\Delta q) = y_3(q) + 2r_2 y_1(q) - 2r_1 y_2(q) \tag{A-8}$$

Thus,

$$\frac{\partial y_1}{\partial r_1} = 0, \quad \frac{\partial y_1}{\partial r_2} = -2y_3, \quad \frac{\partial y_1}{\partial r_3} = 2y_2 \tag{A-9}$$

or, equivalently:

$$X_1 = \frac{\partial y_1}{\partial r} = \begin{bmatrix} 0 & -2y_3 & 2y_2 \end{bmatrix} \tag{A-10}$$

Similarly,

$$X_2 = \frac{\partial y_2}{\partial r} = \begin{bmatrix} 2y_3 & 0 & -2y_1 \end{bmatrix} \tag{A-11}$$

and

$$X_3 = \frac{\partial y_3}{\partial r} = \begin{bmatrix} -2y_2 & 2y_1 & 0 \end{bmatrix} \tag{A-12}$$

Clearly, the same line of reasoning applies to $\vec{y}_n(q)$ as given by Eq. (2) with the result that:

$$X = \begin{bmatrix} 0 & -2y_3 & 2y_2 \\ 2y_3 & 0 & -2y_1 \\ -2y_2 & 2y_1 & 0 \\ 0 & -2y_6 & 2y_5 \\ 2y_6 & 0 & -2y_4 \\ -2y_5 & 2y_4 & 0 \end{bmatrix} \tag{A-13}$$

Appendix B: Incremental Rotation Quaternions

It is well known that a general unit quaternion can be written in the form (Kuipers 1999)

$$q = \cos(\theta/2) + \vec{u}\sin(\theta/2) \tag{B-1}$$

where $\vec{u}$ is a unit vector specifying an axis of rotation and θ is the angle of rotation about that axis. Using such a quaternion, if v is any three-dimensional vector (quaternion

with zero real part) expressed in a fixed reference (or "world") coordinate system, then the rotation of v into a new vector, v', is accomplished by

$$v' = qvq^{-1} \tag{B-2}$$

In this expression, v' is also expressed in the fixed reference coordinate system and from (Kuipers 1999), for any non-zero quaternion p, regardless of its magnitude, the quaternion inverse is defined by

$$p^{-1} = \frac{\begin{pmatrix} p_0 & -p_1 & -p_2 & -p_3 \end{pmatrix}}{|p|^2} = \frac{p^*}{|p|^2} \tag{B-3}$$

It is important to recognize that Eq. (B-2) is valid even for quaternions that are not of unit magnitude. To see that this is so, suppose α is any real number other than zero, and that q is a unit quaternion as defined by Eq. (B-1). Now let

$$p = \alpha q \tag{B-4}$$

and let v' be redefined as

$$v' = pvp^{-1} \tag{B-5}$$

Evidently, from Eq. (B-3)

$$p^{-1} = \frac{p^*}{|p|^2} = \frac{\alpha q^*}{\alpha^2} = \frac{q^{-1}}{\alpha} \tag{B-6}$$

so Eq. (B-5) becomes

$$v' = \alpha qv\left(\frac{q^{-1}}{\alpha}\right) = qvq^{-1} \tag{B-7}$$

This is the same result as Eq. (B-2), which establishes the fact that any non-zero quaternion can be multiplied by any non-zero scalar without changing the rotation accomplished by the given quaternion.

Granted the result of Eq. (B-7), suppose q is a unit quaternion as defined by Eq. (B-1), and suppose further that the rotation angle, θ, is limited to the range: $-\pi < \theta < \pi$. This limitation defines the term *incremental rotation quaternion* used elsewhere in this paper. Obviously, such quaternions are not capable of accomplishing arbitrary rotations by a single application of Eq. (B-2). However, it is equally clear that a succession of incremental rotations can realize any rotation, without any limitation on the range of θ. It is for this reason that the term "incremental" is introduced here to characterize such quaternions.

From the above definition, for an incremental rotation quaternion, evidently $\cos\left(\frac{\theta}{2}\right) > 0$. Thus, any such unit quaternion can be transformed to *unit real* form by dividing it by its real part with the result

$$p = \frac{q}{\cos\left(\theta/2\right)} = \left(1, \quad \vec{u}\tan\left(\theta/2\right)\right) = \left(1, \quad \vec{r}\right) \tag{B-8}$$

The importance of this new "canonical form" (applicable only to incremental rotation quaternions) is that it has only three variable components rather than the four of a unit quaternion, and therefore reduces the order of the Gauss-Newton iteration problem as explained in Section 5 of this paper. Fortuitously, and unexpectedly, another advantage of using incremental rotation quaternions in Gauss-Newton iteration is that the X matrix used in this method, and derived in Appendix A above, is thereby greatly simplified. In this regard, it is important to realize that the limitation of rotation angles associated with unit real quaternions is of no consequence in Gauss-Newton iteration since multiple iterations allow any quaternion to be achieved by this estimation process.
As a final remark, the derivation of the X matrix in Appendix A assumes that $\hat{q}$ is a unit quaternion. Thus, in Figure 1 the results of Eq. (20) are normalized to obtain a unit quaternion after every iteration step. This means that there are no limitations on the magnitude of $\vec{r}$ when unit real quaternions are used for iteration in the indicated way.

Appendix C: LISP Simulation Code

simplified-gauss-newton iteration.cl

```
;This code written in ANSII Common Lisp by Prof. Robert B. McGhee (mcghee@
;cs.nps.navy.mil) at the Naval Postgraduate School, Monterey, CA93940.
;Date of latest update: 14 September 01.

(load "c:\\my documents\\papers\\lisp code for gn filter\\quaternion-functions")

(defvar b (list 0 (- (cos (deg-to-rad 60))) 0 (sin (deg-to-rad 60))))

(defun 3-q-prod (q1 q2 q3)
 (quaternion-product q1 (quaternion-product q2 q3)))

(defun X-matrix (q)
  (let* ((2y (scalar-multiply 2 (computed-measurement q))) (2u1 (first 2y))
         (2u2 (second 2y)) (2u3 (third 2y)) (2v1 (fourth 2y)) (2v2 (fifth 2y))
          (2v3 (sixth 2y)) (Xcol1 (list 0 2u3 (- 2u2) 0 2v3 (- 2v2)))
          (Xcol2 (list (- 2u3) 0 2u1 (- 2v3) 0 2v1))
          (Xcol3 (list 2u2 (- 2u1) 0 2v2 (- 2v1) 0)))
        (transpose (list Xcol1 Xcol2 Xcol3))))

(defun computed-measurement (q)
  (let* ((q-inv (quaternion-inverse q))
         (q1 (3-q-prod q-inv k q)) (q2 (3-q-prod q-inv b q)))
        (append (rest q1) (rest q2))))
```

```
(defun normalize-measurement (y)
  (append (normalize-vector (firstn 3 y)) (normalize-vector (nthcdr 3 y))))

(defun random-start (q-true max-start-error)
  (normalize (vector-add q-true (scalar-multiply max-start-error
                                  (list (- (random 2.0) 1.0) (- (random 2.0) 1.0)
                                        (- (random 2.0) 1.0) (- (random 2.0) 1.0))))))

(defun noise-vector (standard-deviation)
  (do* ((i 5 (1- i))
        (noise (list (gaussian-noise standard-deviation))
               (cons (gaussian-noise standard-deviation) noise)))
       ((zerop i) noise)))

(defun gaussian-noise (standard-deviation)
  (do* ((i 11 (1- i))
        (noise (- (random 1.0) .5) (+ (- (random 1.0) .5) noise)))
       ((zerop i) (* standard-deviation noise))))

(defun noisy-measurement (q standard-deviation)
  (normalize-measurement (vector-add (computed-measurement q)
                                     (noise-vector standard-deviation))))

(defun rotation-vector (measurement-vector estimated-q)
  (let* ((q estimated-q) (y0 measurement-vector) (y (computed-measurement q))
         (error (vector-subtract y0 y)) (X (X-matrix q))
         (X-trans (transpose X))
         (M (matrix-inverse (matrix-multiply X-trans X)))
         (N (matrix-multiply M X-trans)))
        (post-multiply N error)))

(defun best-q-sequence (q-true max-start-error sigma-noise number-of-GN-
cycles)
  (do* ((measurement (noisy-measurement q-true sigma-noise))
        (q-start (random-start q-true max-start-error))
        (q-cap q-start)
        (count (1- number-of-GN-cycles)(1- count))
        (r (rotation-vector measurement q-cap) (rotation-vector measurement q-
cap))
        (delta-q (quaternion-product q-cap (cons 0 r))
                 (quaternion-product q-cap (cons 0 r)))
        (q-cap (normalize (vector-add q-cap delta-q))
               (normalize (vector-add q-cap delta-q)))
        (estimation-sequence (list q-cap q-start) (cons q-cap estimation-sequence)))
       ((zerop count) (cons q-true estimation-sequence))))
```

```
(defun random-q () (normalize (list (- (random 2.0) 1) (- (random 2.0) 1)
                                    (- (random 2.0) 1) (- (random 2.0) 1))))

(defun error-sequence (q-true max-start-error sigma-noise number-of-cycles)
  (do* ((q-seq (best-q-sequence q-true max-start-error sigma-noise number-of-
cycles))
        (q-true (first q-seq))
        (q-cap-list (rest q-seq) (rest q-cap-list))
        (error-list (list (vector-subtract (first q-cap-list) q-true))
                    (cons (vector-subtract (first q-cap-list) q-true) error-list)))
       ((null (rest q-cap-list))  error-list)))

(defun list-of-error-sequences (max-start-error sigma-noise GN-depth
                                                number-of-sequences)
  (do* ((n (1- number-of-sequences) (1- n))
        (q-true (random-q) (random-q))
        (sequence (list (error-sequence q-true max-start-error sigma-noise GN-
depth))
                  (cons (error-sequence q-true max-start-error sigma-noise GN-depth)
                        sequence)))
       ((zerop n) sequence)))


(defun rms-error-sequence (max-start-error sigma-noise GN-depth number-of-
sequences)
  (do* ((mse max-start-error) (sgmn sigma-noise) (dpth GN-depth)
        (nseq number-of-sequences)
        (count (1- nseq) (1- count))
        (error-sequences-list (list-of-error-sequences mse sgmn dpth nseq)
                              (rest error-sequences-list))
        (error-sequence (first error-sequences-list) (first error-sequences-list))
        (sum-squared-error-sequence (squared-error-sequence error-sequence)
                                    (vector-add sum-squared-error-sequence
                                                (squared-error-sequence error-sequence))))
       ((zerop count)
        (list-sqrt (scalar-multiply (/ 1 nseq) sum-squared-error-sequence)))))

(defun squared-length (vector) (dot-product vector vector))

(defun list-sqrt (list)
  (if list (cons (sqrt (first list)) (list-sqrt (rest list)))))
```

```
(defun squared-error-sequence (error-sequence)
  (do* ((sequence error-sequence (rest sequence))
        (error-quaternion (first sequence) (first sequence))
        (squared-sequence (list (squared-length error-quaternion))
                          (cons (squared-length error-quaternion) squared-sequence)))
       ((null (rest sequence)) (reverse squared-sequence))))

(defun t0 () (reverse (best-q-sequence (random-q) 0.1 0.0 3)))

(defun t1 () (reverse (best-q-sequence (random-q) 0.1 0.01 3)))

(defun t2 () (rms-error-sequence 0.1 0.0 4 1000))

(defun t3 () (rms-error-sequence 0.1 0.01 4 1000))

(defun t4 () (rms-error-sequence 0.1 0.03 4 1000))

(defun t5 () (rms-error-sequence 0.1 0.1 4 1000))
```

quaternion-functions.cl

```
;This code written in ANSI Common Lisp (Allegro CL 5.0) by Prof. Robert
;McGhee at the Naval Postgraduate School, Monterey, CA93943. Contact:
;mcghee@cs.nps.navy.mil. Date of last modification: May 9, 2000.

(load "c:\\my documents\\papers\\lisp code for gn filter\\robot-kinematics")

(defun quaternion-product (Q Q1)
  (let ((w (first Q)) (x (second Q)) (y (third Q)) (z (fourth Q))
        (w1 (first Q1)) (x1 (second Q1)) (y1 (third Q1)) (z1 (fourth Q1)))
       (list (- (* w w1) (* x x1) (* y y1) (* z z1))
             (+ (* x w1) (* w x1) (- (* z y1)) (* y z1))
             (+ (* y w1) (* z x1) (* w y1) (- (* x z1)))
             (+ (* z w1) (- (* y x1)) (* x y1) (* w z1)))))

(defun quaternion-conjugate (Q)
  (list (first Q) (- (second Q)) (- (third Q)) (- (fourth Q))))

(defun quaternion-inverse (Q)
  (scalar-multiply (/ 1 (dot-product Q Q)) (quaternion-conjugate Q)))

(defun rotate-vector (quaternion vector) ;Vector is quaternion with leading
  (let* ((q quaternion) (v vector)       ;element zero.
         (q-inv (quaternion-inverse q)))
        (quaternion-product q (quaternion-product v q-inv))))
```

```
(defun quaternion-i (angle)
  (list (cos (* .5 angle)) (sin (* .5 angle)) 0 0))

(defun quaternion-j (angle)
  (list (cos (* .5 angle)) 0 (sin (* .5 angle)) 0))

(defun quaternion-k (angle)
  (list (cos (* .5 angle)) 0 0 (sin (* .5 angle))))

(defun euler-to-quaternion (azimuth elevation roll)
  (quaternion-product (quaternion-k azimuth)
                      (quaternion-product (quaternion-j elevation)
                                          (quaternion-i roll))))

(defun unit-quaternion (angle axis)
  (cons (cos angle) (scalar-multiply (sin angle) axis)))

(defun quaternion-rotation (unit-quaternion)
  (transpose (list (rest (rotate-vector unit-quaternion '(0 1 0 0)))
                   (rest (rotate-vector unit-quaternion '(0 0 1 0)))
                   (rest (rotate-vector unit-quaternion '(0 0 0 1))))))

(defun quaternion-derivative (quaternion pqr)
  (scalar-multiply .5 (quaternion-product quaternion (cons 0 pqr))))

(defun quaternion-homogeneous-transform (quaternion position)
  (let* ((matrix (quaternion-rotation quaternion)))
        (append (concat-matrix matrix (transpose (list position)))
                (list (list 0 0 0 1)))))

(defun normalize (quaternion)
  (scalar-multiply (/ 1 (vector-magnitude quaternion)) quaternion))

(defvar q1 (list (sqrt .5) (sqrt .5) 0 0))

(defvar q2 (list (sqrt .5) 0 (sqrt .5) 0))

(defvar h '(1 0 0 0))

(defvar i '(0 1 0 0))

(defvar j '(0 0 1 0))

(defvar k '(0 0 0 1))
```

robot-kinematics.cl

```
;This code written in ANSI Common Lisp (Allegro CL 5.0) by Prof. Robert
;McGhee at the Naval Postgraduate School, Monterey, CA93943. Contact:
;mcghee@cs.nps.navy.mil. Date of last modification: June 4, 2000.

(defun transpose (matrix)          ;A matrix is a list of row vectors.
  (cond ((null (cdr matrix)) (mapcar 'list (car matrix)))
        (t (mapcar 'cons (car matrix) (transpose (cdr matrix))))))

(defun dot-product (vector-1 vector-2);A vector is a list of numerical atoms.
  (apply '+ (mapcar '* vector-1 vector-2)))

(defun cross-product (vector-1 vector-2)
  (let ((x1 (first vector-1)) (y1 (second vector-1)) (z1 (third vector-1))
        (x2 (first vector-2)) (y2 (second vector-2)) (z2 (third vector-2)))
       (list (- (* y1 z2) (* y2 z1)) (- (* x2 z1) (* x1 z2))
             (- (* x1 y2) (* x2 y1)))))

(defun vector-magnitude (vector) (sqrt (dot-product vector vector)))

(defun normalize-vector (vector)
  (scalar-multiply (/ (vector-magnitude vector)) vector))

(defun post-multiply (matrix vector)
  (cond ((null (rest matrix)) (list (dot-product (first matrix) vector)))
        (t (cons (dot-product (first matrix) vector)
                 (post-multiply (rest matrix) vector)))))

(defun pre-multiply (vector matrix)
  (post-multiply (transpose matrix) vector))

(defun 3D-postmultiply (3D-array vector)
  (if (null (rest 3D-array)) (list (post-multiply (first 3D-array) vector))
      (cons (post-multiply (first 3D-array) vector)
            (3D-postmultiply (rest 3D-array) vector))))

(defun matrix-multiply (matrix1 matrix2)
  (cond ((null (rest matrix1)) (list (pre-multiply (first matrix1) matrix2)))
        (t (cons (pre-multiply (first matrix1) matrix2)
                 (matrix-multiply (rest matrix1) matrix2)))))

(defun chain-multiply (L)     ;L is a list of names of conformable matrices.
  (cond ((null (cddr L)) (matrix-multiply (eval (car L)) (eval (cadr L))))
        (t (matrix-multiply (eval (car L)) (chain-multiply (cdr L))))))
```

```
(defun cycle-left (matrix) (mapcar 'row-cycle-left matrix))

(defun row-cycle-left (row) (append (cdr row) (list (car row))))

(defun cycle-up (matrix) (append (cdr matrix) (list (car matrix))))

(defun unit-vector (one-column length)        ;Column count starts at 1.
  (do ((n length (1- n))
       (vector nil (cons (cond ((= one-column n) 1) (t 0)) vector)))
      ((zerop n) vector)))

(defun unit-matrix (size)
  (do ((row-number size (1- row-number))
       (I nil (cons (unit-vector row-number size) I)))
      ((zerop row-number) I)))

(defun concat-matrix (matrix1 matrix2)
  (if matrix1 (cons (append (first matrix1) (first matrix2))
                    (concat-matrix (rest matrix1) (rest matrix2)))))

(defun augment (matrix)
  (concat-matrix matrix (unit-matrix (length matrix))))

(defun normalize-row (row) (scalar-multiply (/ 1.0 (first row)) row))

(defun scalar-multiply (scalar vector)
  (cond ((null vector) nil)
        (t (cons (* scalar (first vector))
                 (scalar-multiply scalar (rest vector))))))

(defun solve-first-column (matrix)     ;Reduces first column to (1 0 ... 0).
  (do* ((remaining-row-list matrix (rest remaining-row-list))
        (first-row (normalize-row (first matrix)))
        (answer (list first-row)
                (cons (vector-add (first remaining-row-list)
                                  (scalar-multiply (- (caar remaining-row-list))
                                                   first-row)) answer)))
       ((null (rest remaining-row-list)) (reverse answer))))

(defun vector-add (vector-1 vector-2) (mapcar '+ vector-1 vector-2))

(defun vector-subtract (vector-1 vector-2) (mapcar '- vector-1 vector-2))

(defun matrix-subtract (matrix-1 matrix-2)
  (mapcar #'vector-subtract matrix-1 matrix-2))
```

```
(defun subtract-unit-matrix (square-matrix)
  (matrix-subtract square-matrix (unit-matrix (length square-matrix))))

(defun sum-of-elements-squared (matrix)
  (apply '+ (mapcar #'dot-product matrix matrix)))

(defun rms-inverse-error-metric (matrix approximate-inverse-matrix)
  (let* ((M matrix) (M-inv approximate-inverse-matrix) (n (length M))
         (error-matrix (subtract-unit-matrix (matrix-multiply M M-inv)))
         (S (sum-of-elements-squared error-matrix)))
        (/ (sqrt S) n)))

(defun first-square (matrix)  ;Returns leftmost square matrix from argument.
  (do ((size (length matrix))
       (remainder matrix (rest remainder))
       (answer nil (cons (firstn size (first remainder)) answer)))
      ((null remainder) (reverse answer))))

(defun firstn (n list)
  (cond ((zerop n) nil)
        (t (cons (first list) (firstn (1- n) (rest list))))))

(defun pivot-row-firstn (n list)
  (append (pivot-row-first (firstn n list)) (nthcdr n list)))

(defun matrix-inverse (matrix)
  (do* ((M (pivot-row-first (augment matrix))
           (pivot-row-firstn n (cycle-left (cycle-up M))))
        (n (1- (length matrix)) (1- n))
        (exit-flag (= 0 (caar M)) (= 0 (caar M))));Prevents division by zero.
       ((or (minusp n) exit-flag) (if (not exit-flag) (first-square M)))
       (setf M (solve-first-column M))))

(defun pivot-row-first (matrix)  ;This function finds row with largest first
  (cond ((null (cdr matrix)) matrix) ;element and moves it to top of matrix.
        (t (if (> (abs (caar matrix))
                  (abs (caar (pivot-row-first (cdr matrix)))))) matrix
               (append (pivot-row-first (cdr matrix))
                       (list (car matrix)))))))
```

```
(defun dh-matrix (rotate twist length translate)
  (let ((cosrotate (cos rotate)) (sinrotate (sin rotate))
        (costwist (cos twist)) (sintwist (sin twist)))
       (list (list cosrotate (- (* costwist sinrotate))
                   (* sintwist sinrotate) (* length cosrotate))
             (list sinrotate (* costwist cosrotate)
                   (- (* sintwist cosrotate)) (* length sinrotate))
             (list 0. sintwist costwist translate)
             (list 0. 0. 0. 1.))))

(defun homogeneous-transform (orientation position)
  (let* ((roll (first orientation)) (elevation (second orientation))
         (azimuth (third orientation)) (x (first position))
         (y (second position)) (z (third position))
         (spsi (sin azimuth)) (cpsi (cos azimuth)) (sth (sin elevation))
         (cth (cos elevation)) (sphi (sin roll)) (cphi (cos roll)))
        (list (list (* cpsi cth) (- (* cpsi sth sphi) (* spsi cphi))
                    (+ (* cpsi sth cphi) (* spsi sphi)) x)
              (list (* spsi cth) (+ (* cpsi cphi) (* spsi sth sphi))
                    (- (* spsi sth cphi) (* cpsi sphi)) y)
              (list (- sth) (* cth sphi) (* cth cphi) z)
              (list 0. 0. 0. 1.))))

(defun inverse-H (H)        ;H is a 4x4 homogeneous transformation matrix.
  (let* ((minus-P (list (- (fourth (first H)))
                        (- (fourth (second H)))
                        (- (fourth (third H)))))
         (inverse-R (transpose (first-square (reverse (rest (reverse H))))))
         (inverse-P (post-multiply inverse-R minus-P)))
        (append (concat-matrix inverse-R (transpose (list inverse-P)))
                (list (list 0 0 0 1)))))

(defun rotation-matrix (euler-angles)
  (let* ((roll (first euler-angles)) (elevation (second euler-angles))
         (azimuth (third euler-angles))
         (spsi (sin azimuth)) (cpsi (cos azimuth)) (sth (sin elevation))
         (cth (cos elevation)) (sphi (sin roll)) (cphi (cos roll)))
        (list (list (* cpsi cth) (- (* cpsi sth sphi) (* spsi cphi))
                    (+ (* cpsi sth cphi) (* spsi sphi)))
              (list (* spsi cth) (+ (* cpsi cphi) (* spsi sth sphi))
                    (- (* spsi sth cphi) (* cpsi sphi)))
              (list (- sth) (* cth sphi) (* cth cphi)))))
```

```
(defun body-rate-to-euler-rate-matrix (euler-angles)
  (let* ((roll (first euler-angles)) (elevation (second euler-angles))
         (sth (sin elevation)) (cth (cos elevation)) (tth (tan elevation))
         (sphi (sin roll)) (cphi (cos roll)))
        (list (list 1 (* tth sphi) (* tth cphi))
              (list 0 cphi (- sphi))
              (list 0 (/ sphi cth) (/ cphi cth)))))

(defun rad-to-deg (angle) (* 57.29577951308232 angle))

(defun deg-to-rad (angle) (* 0.017453292519943295 angle))

(defvar M '((1 1 -1) (-1 3 -1) (3 -5 -2)))

(defvar N '((1 2 3) (4 5 6) (7 8 9)))

(defvar L '((3 2 1) (4 5 6) (7 8 9)))

(defun test1 () (matrix-inverse M)) ;Problem 2-9(a) in Kuo.

(defun test2 () (matrix-inverse N))

(defun test3 () (matrix-inverse L))

(defun test4 () (matrix-multiply L (test3)))

(defun test5 () (rms-inverse-error-metric L (matrix-inverse L)))

(defun test6 () (rms-inverse-error-metric M (matrix-inverse M)))

(defvar v '(1 2 3))

(defvar 3D-array '(((1 1 1) (2 2 2)) ((3 3 3) (4 4 4))))

(defun test () (3D-postmultiply 3D-array v))
```

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	2
2.	Dudley Knox Library, Code 013 Naval Postgraduate School Monterey, CA 93943-5100	2
3.	Research Office, Code 09 Naval Postgraduate School Monterey, CA 93943-5138	1